Not The Best

The task is to find the **second shortest path** from node 1 to node N in an undirected graph. In this problem, paths may revisit nodes or edges, so the solution must account for paths that might include cycles. Simply looking for two distinct simple paths is insufficient; instead, we must identify the path whose total distance is slightly larger than the optimal path but minimal among all longer alternatives.

A practical way to solve this is by using a modified **Dijkstra's algorithm** that tracks both the shortest and the second shortest distances to every node. During traversal, we relax edges not only from the shortest known path but also from the second shortest. This ensures we explore paths that slightly deviate from the optimal route, including detours or repeated segments, while maintaining efficiency with O((V+E)log⁡V)O((V + E) \log V)O((V+E)logV) complexity. This approach correctly handles scenarios where the second shortest path may reuse nodes or edges instead of being entirely disjoint from the shortest path.

The method can be outlined as follows:

1. **Initialization**
    - Create two arrays: dist1 for the shortest distance and dist2 for the second shortest distance to each node.
    - Initialize all entries to a very large number (infinity).
    - Prepare a min-priority queue to store pairs (distance, node).
2. **Starting Node**
    - Set dist1[1] = 0 for the starting node.
    - Push (0, 1) into the priority queue.
3. **Process Nodes**
    - While the queue is not empty:
        - Extract the node with the smallest distance.
        - If this distance exceeds the node's current second shortest distance, skip it.
        - Otherwise, examine all neighboring nodes.
4. **Relax Edges**

- For each neighbor with edge weight w:
  - Compute newDistance = currentDistance + w.
  - If newDistance is less than the neighbor's shortest distance:
    - Update the neighbor's second shortest distance to the previous shortest value.
    - Update the shortest distance to newDistance.
    - Push (newDistance, neighbor) into the queue.
  - Else if newDistance is between the shortest and second shortest:
    - Update the second shortest distance to newDistance.
    - Push (newDistance, neighbor) into the queue.

5. **Result**
   - After processing all nodes, dist2[N] contains the length of the second shortest path from node 1 to node N.

     Input:

     2

     3 3

     1 2 100

     2 3 200

     1 3 50

     Execution:

     1. Initialize:

     o dist1 = [0, ∞, ∞], dist2 = [∞, ∞, ∞] o Queue: [(0,1)]

     2. Process (0,1):

     o 1→2: new=100 < ∞ → dist1[2]=100, push (100,2) o 1→3:

     new=50 < ∞ → dist1[3]=50, push (50,3)

     o Queue: [(50,3), (100,2)]

3. Process (50,3):

o 3→1: new=100 > 0 but < ∞ → dist2[1]=100, push (100,1) o

3→2: new=250 > 100 but < ∞ → dist2[2]=250, push

(250,2) o Queue: [(100,2), (100,1), (250,2)]

4. Process (100,2):

o 2→1: new=200 > 100 (dist2[1]) → skip o 2→3: new=300 >

50 but < ∞ → dist2[3]=300, push (300,3) o Queue: [(100,1),

(250,2), (300,3)]

5. Process (100,1):

o 1→2: new=200 > 100 but < 250 → dist2[2]=200, push

(200,2) o 1→3: new=150 > 50 but < 300 → dist2[3]=150,

push (150,3) o Queue: [(150,3), (200,2), (250,2), (300,3)]

6. Process (150,3): o Target reached with second shortest = 150

Answer: 150

Output:

```
150
```

Pseudocode: FOR each test case:

```
    READ number of nodes and edges
    BUILD the adjacency list of the graph

    shortest = [INFINITY] * (N + 1)
    secondShortest = [INFINITY] * (N + 1)
    shortest[1] = 0

    minHeap = empty priority queue
```

```
INSERT (0, 1) INTO minHeap


WHILE minHeap is not empty:
    currentDist, node = EXTRACT_MIN from minHeap
    IF currentDist > secondShortest[node]:
        CONTINUE
    FOR each neighbor, weight in adjacency[node]:
        newDist = currentDist + weight
        IF newDist < shortest[neighbor]:
            secondShortest[neighbor] = shortest[neighbor]
            shortest[neighbor] = newDist
            INSERT (newDist, neighbor) INTO minHeap
        ELSE IF shortest[neighbor] < newDist < secondShortest[neighbor]:
            secondShortest[neighbor] = newDist
            INSERT (newDist, neighbor) INTO minHeap


PRINT secondShortest[N]
```

Code: https://github.com/pulokdas062/Graph-Algorithm/edit/main/Dijkstra/Not%20the%20best/not%20the%20best.cpp